

## TITLE

1AP20 RECEIVED 09 FEB 2006

Huffman coding and decoding.

## 5 FIELD OF THE INVENTION

Embodiments of the present invention relate to Huffman encoding and decoding. In particular they relate to improved mechanisms for encoding and decoding and an improved representation of a Huffman tree.

10

## BACKGROUND TO THE INVENTION

In digital processing, if a message comprises a sequence of symbols, each distinct symbol can be represented as a distinct binary codeword. Huffman's algorithm uses a table of the frequencies of occurrence of each symbol in a message and optimizes the variable length codewords such that the most frequent codeword has the shortest length. This results in data compression and Huffman coding is commonly used in audio and video compression coding, for example, MPEG.

20 "A method for the construction of minimum-redundancy codes", by David A Huffman, Proceedings of the IRE 40 (1952) 1098-1101, Introduces Huffman Coding.

If there are nine symbols  $S_0, S_1 \dots S_8$  with the following respective frequencies of occurrence 5, 5, 6, 1, 2, 3, 16, 9, 9, then they can be encoded using the Huffman algorithm into the binary tree illustrated in Fig. 1.

The tree 10 comprises leaf nodes  $S_i$  and interior nodes  $F_i$  arranged in  $H$  levels. Each leaf node depends from a single interior node on the next lowest level and represents a symbol. Each interior node depends from a single interior node on the next lowest level.

30 The level  $L$  of a node is defined by setting the root to level 0, and the other nodes have a level that is one higher than the level of the node from which it depends. The highest level is the height  $H$  of the Huffman tree. The symbols (i.e. the leaves of  $T$ ) are labelled from left to right as  $S_0, S_1, S_2 \dots S_8$ .

35 The Huffman tree illustrated in Fig. 1 results in the following coding of the symbols

Symbol	Codeword
S0	000
S1	001
S2	010
S3	01100
S4	01101
S5	0111
S6	10
S7	110
S8	111

Table 1

- 5 In its simplest representation, a Huffman binary tree of height H may be represented using a word for each node of the tree. The size of such a representation makes it difficult to search during decoding.
- 10 "Memory efficient and high speed search Huffman coding ", by Hashemian, IEEE Trans on Comms, Vol. 43, No 10, 2576- , Oct 1995, reduces the storage space required to represent a Huffman tree and increases the decode speed using the tree. A sparse single-side growing Huffman tree is created and partitioned into smaller and less sparse clusters (sub-trees), each L levels apart. A super-tree is constructed in which each cluster is represented by a node. A super table specifies the super-tree. It specifies, for
- 15 each node, the length of the cluster associated with that node and the address of the look up table for that cluster. A negative entry in the look-up table is a reference back to the super-table. A positive entry indicates that a symbol has been found and the magnitude of the entry provides the location of the symbol in memory, the codeword and the codeword length.
- 20 In "A memory-efficient and fast Huffman decoding algorithm", by Chen, Inform Process Lett. 69 (1999) 119-122, weight is given to a leaf node equal to the number of leaves in a complete tree under the node. It is dependent upon the level of the node within the tree. Every leaf node is assigned a number equal to the cumulative weight of all the
- 25 leaves appearing before it and its own weight. A codeword is given an equivalent

cumulative weight as if it were a node on a tree. The actual cumulative weights are searched to determine if one matches the equivalent cumulative weight. If there is a match and the weights of the matching nodes are the same, then the codeword is a leaf node of the tree (i.e. a symbol).

5

"An efficient decoding technique for Huffman codes", Chowdhury et al, Inform Process Lett, 81 (2002), 305-308., truncates a Huffman tree by removing all the leaves to improve memory use and search speed.

- 10 It would be desirable to provide an alternative representation of a Huffman binary tree and an improved Huffman decoding mechanism for decoding a received string of binary digits.

## 15 BRIEF DESCRIPTION OF THE INVENTION

According to one embodiment of the invention there is provided a method of decoding a bitstream encoded according to a Huffman coding tree of height H comprising:

extracting a first codeword of H bits from the bitstream;

- 20 modifying the codeword by shifting it by a first shift value;

using this modified codeword to identify using at least a first data structure either a symbol or a second data structure having an associated second offset value and an associated second shift value; and

if a second data structure is identified using the first data structure:

- 25 modifying the codeword by subtracting the second offset value and shifting the result by the second shift value; and

using this modified codeword to identify using the second data structure either a symbol or a third data structure having an associated third offset value and an associated third shift value.

30

According to another embodiment of the invention there is provided a method of decoding a bitstream encoded according to a Huffman coding tree of height H comprising:

extracting a codeword of H bits from the bitstream;

- 35 shifting the codeword by a predetermined shift value; and

using the modified codeword to identify a symbol using at least a first data structure.

According to another embodiment of the invention there is provided a decoder for decoding a bitstream encoded according to a Huffman coding tree of height H

5 comprising:

a memory for storing a plurality of data structures representing the Huffman coding tree of height H including at least a first data structure having an associated first offset value and an associated first shift value and a second data structure having an associated second offset value and an associated second shift value; and

10 a processor operable to

subtract a current offset value from a codeword of H bits taken from the bitstream;  
shift the result by the associated shift value; and  
address the associated data structure using the result.

15 According to another embodiment of the invention there is provided a method of decoding a bitstream encoded according to a Huffman coding tree of height H comprising:

storing a first data structure comprising a value for each possible node at a first level of the tree;

20 storing a second data structure comprising a value for each possible node within a first sub-tree at a second, lower level of the tree;

extracting a first codeword of H bits from the bitstream;

converting the value of the first codeword into a first node position within the tree at the first level of the tree; and

25 accessing the first data structure to obtain the value corresponding to the first node position, wherein that value refers to the second data structure;

converting the value of the first codeword into a second node position within the first sub-tree at the second level of the tree; and

accessing the second data structure to obtain the value corresponding to the second

30 node position.

According to another embodiment of the invention there is provided a method of decoding a codeword from a bit stream comprising:

receiving a representation of a Huffman tree as a plurality of ordered data structures

- comprising: a first data structure associated with an identified first level L1 of the tree and comprising a plurality of data entries, each entry corresponding to a node of a full tree at the identified first level and at least a second data structure associated with an identified second level L2 of the tree and with an identified first sub-tree and comprising
- 5 a plurality of data entries, each entry corresponding to a node of the first sub tree, when full, at the second identified level;
- obtaining a value for a first level L1 in a Huffman tree
- identifying the node in the first level L1 of the tree, when full, corresponding to the first L1 bits of the codeword;
- 10 obtaining from the first data structure a data entry for the identified node, that identifies a further data structure if the identified node is an interior node and otherwise identifies a symbol; and
- if the identified node is an interior node:
- obtaining a value for a second level L2 in a Huffman tree, being a higher level
- 15 than the first level L1;
- obtaining a value identifying a first sub-tree;
- identifying the node in the second level L2 of the first sub-tree, when full, corresponding to the first L2 bits of the received bit stream;
- obtaining from a further data structure a data entry for the identified node, that
- 20 identifies a further data structure if the identified node is an interior node and otherwise identifies a symbol.

- According to another embodiment of the invention there is provided data representing a Huffman coding tree comprising leaf nodes and interior nodes arranged in H levels,
- 25 wherein each leaf node depends from a single interior node on the next lowest level and represents a symbol and each interior node depends from a single interior node on the next lowest level, the data comprising:
- a first data structure identifying, for each of the nodes within a first specified level of the tree, a symbol for each leaf node and a further data structure for each interior node,
- 30 including a second data structure for a first interior node;
- at least a second data structure, identified by the first data structure, identifying for each of the nodes within a sub-tree, depending from the first interior node, and at a second specified level of the tree, a symbol for each leaf node and a further data structure for an interior node, if any; and
- 35 data specifying at least the first level, the second level and the first interior node.

According to another embodiment of the invention there is provided a method of representing a Huffman binary tree comprising:

producing a first data structure associated with an identified first level L1 of the tree and

5 comprising a plurality of data entries, each entry corresponding to a node of a full tree at the identified first level and identifying a further data structure if that node is an interior node and otherwise identifying a symbol; and

producing at least a further data structure associated with an identified second level L2 of the tree and with an identified first sub-tree and comprising a plurality of data entries,

10 each entry corresponding to a node of the first sub tree, when full, at the second identified level L2 and identifying a further data structure if that node is an interior node and otherwise identifying a symbol.

One advantage associated with embodiments of the invention are that the technique  
15 utilizes the probability of occurrence of codewords. The more frequently occurring codewords will be decoded first.

Another advantage associated with embodiments of the invention is the low processing overhead associated with decoding- in each decoding iteration there is only a maximum  
20 of one subtraction, one shift and one compare. The decoding can consequently be implemented very efficiently in a digital signal processor (DSP).

Another advantage associated with embodiments of the invention is that the maximum search can be designed to be less than  $\log_2 N$

25

#### BRIEF DESCRIPTION OF THE DRAWINGS

For a better understanding of embodiments of the present invention and to understand  
30 how the same may be brought into effect, reference will now be made by way of example only to the accompanying drawings in which:

Fig. 1 illustrates a Huffman binary tree;

Fig. 2 illustrates a device that is usable as an encoder and/or decoder; and

35 Fig. 3 illustrates the decoding process;

Fig. 4 illustrates another Huffman tree.

# DETAILED DESCRIPTION OF EMBODIMENTS OF THE INVENTION

## Coding

In this example, let there be nine symbols S0, S1 ...S8 with the following respective frequencies of occurrence 5, 5, 6, 1, 2, 3, 16, 9, 9. They can be encoded using the Huffman algorithm into the binary tree illustrated in Fig. 1.

This Huffman tree can be represented as an ordered sequence of data structures as defined below in the following look-up tables (Tables 2, 3 and 4).

Lookup table	Offset_value	Shift_value	Sub_table
Lookup table 0	0	2	0
Lookup table 1	12	0	1

Table 2

sub-table 0								
y	0	1	2	3	4	5	6	7
symbol_locate	->S0	->S1	->S2	->sub-table 1	->S6	->S6	->S7	->S8

Table 3

sub table 1				
y	0	1	2	3
symbol locate	-> S3	-> S4	-> S5	->S5

Table 4

The first data structure (Table 2) is a first look-up table. It identifies an offset\_value and shift\_value pair to be used with each identified sub-table. In this example, it designates that a zero offset and shift\_value of 2 should be used with sub-table 0 (Table 3) and that an offset\_value of 12 and a shift\_value of 0 should be used with sub-table 1 (Table 4).

5

The second data structure (Table 3) is a look-up table. It is addressed using a value y and provides in return a value for symbol\_locate. The value for symbol\_locate may either point to a symbol value or point to another look-up table. For example, the values for y equal to 0, 1, 2, 4, 5, 6, 7 respectively point to the symbols S0, S1, S2, S6, S6, S7, S8 and the value for y equal to 3 points to sub-table 1 (Table 4).

10

The third data structure (Table 4) is a look-up table. It is addressed using a value y and provides in return a value for symbol\_locate. The value for symbol\_locate may either point to a symbol value or point to another look-up table. In this example the values for y equal to 0, 1, 2, 3 respectively point to the symbols S3, S4, S5, S5.

15

The 8-bit symbol\_locate may be formatted so that its most significant bit (MSB) identifies whether it is a pointer to a symbol (MSB=0) or a pointer to another look-up table (MSB=1). Bits 1-7 gives an address of the correct symbol or look-up table. For example, the value of symbol\_locate in sub-table0 at y=3 would be 1000 0001 indicating sub-table 1. The value of symbol\_locate in sub-table0 at y=1 would be 0000 0000 indicating symbol S0 according to Table 5 below.

20

symbol locate	Symbol
0000 0000	S0
0000 0001	S1
0000 0010	S2
0000 0011	S3
0000 0100	S4
0000 0101	S5
0000 0110	S6
0000 0111	S7
0000 1000	S8



Table 5

The creation of the above described data structures will now be described. Table 6  
 5 illustrates the methodology used to create the data structures.

In Table 6, column 1 lists in order the symbols  $S_i$  for  $i=0, 1, 2 \dots N-1$ . Column 2 list  
 adjacent each symbol its codeword. Column 3 gives the count value  $\text{Count}_{S_i}$  for each  
 codeword.

10

$\text{Count}_{S_i}$  is the value of the sequence of bits obtained by adding 1's to the codeword for  
 symbol  $S_i$  until it has a length  $H$ . For example, the count value for  $S_0$  is the value of  
 00011 (2+1), the count value of  $S_2$  is the value of 01011 (8+2+1). Alternatively the  
 count value  $\text{Count}_{S_i}$  may be calculated, for each symbol  $S_i$  occupying a level  $L_i$  and a  
 15 codeword of value  $V$ , as  $[(V+1)*2^{(H-L_i)}] - 1$ . E.g. for  $S_3$  {01100},  $V=12$  &  $L=0$ , and  
 count =  $[(12+1)*(2^0)] - 1 = 12$  and for  $S_8$  {111},  $V=7$  &  $L=2$ , and count =  $[(7+1)*(2^2)] - 1 =$   
 31.

20

Column 4 gives a factor for each symbol. This is calculated, for symbol  $S_i$ , as  $\text{count}_{S_i} -$   
 $\text{count}_{S_{i-1}}$ , where  $\text{count}_{S_{-1}} = -1$ . It may alternatively be calculated, for Symbol  $S_i$  at level  
 $L_i$ , as  $2^{(H-L_i)}$ .

Column 5 gives the 'shift' for each symbol. It is the logarithm to the base 2 of the  
 symbol's factor. The shift may alternatively be calculated, for each symbol  $S_i$ , as  $L_i$ .

1	2	3	4	5	6	7	8	9
Si	Codeword	Count <sub>Si</sub>	Factor $2^{(H-L_i)}$	Shift $H-L_i$	$y = (x - 0) \gg 2$	$y = (x - 12) \gg 0$		
S-1		-1						
S0	000	3 (011)	4	2	0 (0)	-		
S1	001	7 (111)	4	2	1 (1)	-		
S2	010	11 (1011)	4	2	2 (10)	-		
S3	01100	12 (1100)	1	0	3 (11)	0		
S4	01101	13 (1101)	1	0	3 (11)	1		
S5	0111	15 (1111)	2	1	3 (11)	3		
S6	10	23 (10111)	8	3	5 (101)	-		
S7	110	27 (11011)	4	2	6 (110)	-		
S8	111	31 (11111)	4	2	7 (111)	-		

Table 6

- 5 Equation 1 is used to calculate the values in columns 6 and 7, which provide the content for the second and third data structures (Tables 3 and 4).

$$y = (x - \text{offset\_value}) \gg (\text{shift\_value}) \quad \text{-----Equation 1}$$

- 10 where

x is the value of a codeword, offset-value is an integer greater than or equal to 0, and shift value is an integer greater than or equal to 0. The operator ">>" indicates that the value  $(x - \text{offset\_value})$  should be truncated by shift\_value, losing shift\_value of its least significant bits. This shift operation to the left is equivalent to a divide by  $2^{\text{shift\_value}}$

- 15 operation but is easily performed by a digital signal processor (DSP).

For column 6 i.e. the second data structure (Table 3) the offset\_value is set to zero and

the shift\_value is set to 2. The equation  $y = (x - 0) \gg 2$  is used to calculate each of the entries in column 6, which correspond to y values in sub-table 0 (Table 3). The symbols S3, S4 and S5 share a common y value. A further sub-table is used for these symbols.

- 5 For column 7 i.e. the third data structure (Table 4) the offset\_value is 12 and the shift\_value is set to 0. The equation  $y = (x - 12) \gg 0$  is used to calculate each of the entries in column 7 for unresolved symbols S3, S4 and S5, which correspond to the y values in sub-table 1 (Table 4).
- 10 Consequently, the sub-table 0 represents the nodes in the Huffman binary tree of Fig. 1 at a level  $L_i$  that is shift\_value levels removed from the top of the tree i.e.  $L_i = H - \text{shift\_value}$ .

- The nodes at  $L=3$ , in order are, S0, S1, S2, F2, empty, empty, S7, S8. The empty nodes  
 15 would depend from a leaf node associated with symbol S6. The symbol S6 is therefore placed in the dependant empty nodes. The nodes at  $L=3$ , can be re-written as S0, S1, S2, F2, S6, S6, S7, S8. This corresponds to the content of the entries in sub-table 0.

- The internal node F2, is the root of a sub-tree. The sub-table 1 represents this sub-tree.  
 20 The count value of any leaf nodes that depend from F2 will have values between 12 and 15. The count values of the codewords are renormalized by subtracting 12, the offset value for sub-table 1.

- The sub-table 1 represents the nodes of the sub-tree at a level L in the Huffman binary  
 25 tree equal to  $H - \text{shift\_value}$ . Shift\_value, for sub-table 1, is zero for this sub-tree so  $L=5$ .

- The nodes at  $L=5$  are, in order, S3, S4, empty, empty. The empty nodes would depend  
 from a leaf node associated with symbol S5. The symbol S5 is therefore placed in the dependant empty nodes. The nodes at  $L=5$  can be rewritten as S3, S4, S5, S5. This  
 30 corresponds to the content of the entries in sub-table 1.

The Huffman tree in Fig. 1 is therefore represented by a first data structure, sub-table 0, representing the tree at level  $L=3$  and by a second data structure, sub-table 1, representing a sub-tree at level  $L=5$ .

Sectioning the tree is necessary to allow the most probable symbols to be found first in an efficient manner. However, the introduction of too many sections uses additional memory to store the additional sub-table and additional shift and offset values and requires additional processing steps to decode the less commonly occurring symbols.

5

For example, it is possible to represent the Huffman tree by sectioning it at different levels than  $L=3$  and  $L=5$ . For example, if the tree is sectioned at  $L=3$ ,  $L=4$  and  $L=5$ , there will be three sub-tables. The first entry of the sub-table for  $L=4$  will refer to the sub-table for  $L=5$  and the second entry will refer to  $S_5$ . The first entry of the sub-table for  $L=5$  will refer to  $S_3$  and the second entry will refer to  $S_4$ . The offset for both these sub-table is 12. The shift for the sub-table for  $L=4$  is 1 and the shift for the sub-table for  $L=5$  is 0.

10

The following should be noted for the sectioning of the Huffman tree of Fig. 1 that gave the data structures of Tables 2, 3, 4 and 5.

15

A data structure representing a level  $L$  of a tree (i.e. a tree sectioned at  $L$ ) has a shift value equal to  $H-L$ . A data structure referenced from a node  $N$  in the tree, where the node  $N$  has a codeword value  $V$  and is in level  $L$ , has an offset\_value  $V * 2^{(H-L)}$ .

20

For the sub-table0, the shift value is set equivalent to  $H-L^*$ , where  $L^*$  is the level at which the tree is sectioned. Here it is the lowest level on which multiple leaves appear. In Fig. 1 there is one leaf at  $L=2$ , but multiple leaves at  $L=3$ . Therefore the tree is sectioned at  $L=3$  and the shift\_value = 2. The offset\_value is 0.

25

For example, the sub-table 1 is referenced via node {011} at  $L=3$ .  $V=3$ , therefore the offset\_value for the sub-table 1 is  $3*2^2$ . The shift value for the sub-table1 is set equivalent to  $H-L^*$ , where  $L^*$  is the level at which the tree is sectioned. Here, it is the lowest level on which multiple leaves appear in the sub-tree. In Fig. 1 the sub-tree has as its root  $F_2$  in level  $L=3$ . This sub-tree has one leaf at  $L=4$ , but two at  $L=5$ . Therefore the tree is sectioned at  $L=5$  and shift\_value = 0.

30

Fig. 2 illustrates a device 20 that is operable as an encoder. The encoder device comprises a processor 22 and a memory 24. The memory 24 stores computer program instructions that control the processor 22 to carry-out the encoding process described above.

35

The computer program instructions may be transferred to the memory via a suitable storage medium in which the instructions are embodied or they may be transferred via electrical signals embodying the instructions.

5

The processor 22 has an input 21 for receiving data for encoding and has an output 23 for providing encoded data. The encoded data includes a plurality of data structures representing a Huffman tree and a value for the height H of the tree.

10

### Decoding

Fig 2. illustrates a device 20 operable as a decoder. The decoder comprises a processor 22 and a memory 24. The memory stores data structures representing a Huffman tree as described above. For example the memory 24 may store tables corresponding to those illustrated in Tables 2, 3, 4 and 5 above. The memory also stores computer program instructions that control the processor 22 to carry-out the decoding process described below with reference to Fig. 3.

The computer program instructions may be transferred to the memory via a suitable storage medium in which the instructions are embodied or they may be transferred via electrical signals embodying the instructions.

The processor 22 has an input 21 for receiving data for decoding and has an output 23 for providing decoded data. The height of the Huffman tree is known in the decoder. This may be communicated to the decoder in a header provided with the data for decoding.

30

The decoder performs the following steps as illustrated in Fig. 3 :

At step 100 the current sub-table identifier z is set to zero.

Then at step 102, H bits are extracted from the bitstream, having a value x

Then at step 104, the look-up table is accessed. The offset\_value and shift\_value for sub-table z is obtained.

Then at step 106, calculate a y value for the current sub-table from equation 1 using x  
5 and the obtained values of offset\_value and shift\_value for the current sub-table.

Then at step 108, access the current sub-table, sub-table z. Obtain the symbol\_locate from sub-table z corresponding to the calculated y value.

10 Then at step 110, a comparison is performed on the MSB of the obtained symbol\_locate.  
If the MSB of the obtained symbol\_locate is 0 then the process jumps to step 114  
If the MSB of the obtained symbol\_locate is 1, the process jumps to step 112.

At step 112, the current sub-table identifier z is set equal to the remaining least  
15 significant bits of the symbol\_locate obtained at step 108. Then the process returns to step 104.

At step 114, the symbol corresponding to by the symbol\_locate obtained at step 108.

20 An example of the process illustrated will now be given using the example look-up tables given above (Tables 2, 3, 4 and 5). The codeword to be searched is {0, 1, 1, 1}. Let the extracted bits be {0, 1, 1, 1, 0}.

At step 100, set the current sub-table indicator z = 0  
25

At step 102, extract bits{0, 1, 1, 1, 0}. Therefore x = 14.

At step 104, access the look-up table (Table 2). The offset\_value for sub-table 0 = 0  
and the shift\_value for sub-table 0 = 2.  
30

At step 106, calculate y using equation 1.  
 $y = (x - 0) \gg 2 = 3$

At step 108, access the current sub-table, sub-table 0 (Table 3). Obtain the symbol\_locate from sub-table 0 corresponding to the calculated y value. The symbol\_locate is {1000 0001}.

- 5 At step 110, the MSB of the obtained symbol\_locate is determined to be 1, therefore go to step 112.

At step 112, z is set equal to the remaining least significant bits {000 0001} of the obtained symbol\_locate i.e.  $z=1$ .

10

At step 104, access the current look-up table (Table 2) . Obtain a new offset\_value and a new shift\_value for the new current sub-table (sub-table 1). The offset\_value for sub-table 1 = 12 and the shift\_value for sub-table 1 = 0.

- 15 At step 106, calculate y for current sub-table, sub-table 1, using equation 1.  
 $y = (x - 12) \gg 0 = 2$

At step 108, access the current sub-table (sub-table 1). Obtain the symbol\_locate corresponding to the new y value ( $y=2$ ). The symbol\_locate is {0000 0101}.

20

At step 110, the MSB of the obtained symbol\_locate is determined to be 0, therefore go to step 114.

At step 114, obtain the symbol corresponding to the obtained symbol\_locate.

- 25 The symbol corresponding to the obtained symbol\_locate {0000 0101} is, from Table 5, S5.

It will be understood that the same result will be achieved if the extracted bits are {0, 1, 1, 1, 1} instead of {0, 1, 1, 1, 0}.

30

#### Further illustration

As a further illustration the encoding of the Huffman tree illustrated in Fig. 4 will now be briefly described.

35

Table 7 is created:

$S_i$	codeword	$count_{S_i}$
S0	000	15
S1	001	31
S2	010	47
S3	01100	51
S4	0110100	52
S5	0110101	53
S6	011011	55
S7	0111	63
S8	10	95
S9	110	111
S10	111	127

5 Sectioning the tree at  $L=3$ , gives  $y = (x-0) \gg 4$

y0	0	1	2	3	4	5	6	7
symbol_locate	S0	S1	S2	Goto table1	S8	S8	S9	S10

Sectioning the tree at  $L=4$ , gives  $y = (x-48) \gg 3$ , where  $V\{011\} * 2^4 = 48$

y1	0	1
symbol_locate	Goto table2	S7

10

Sectioning the tree at  $L=5$ , gives  $y = (x-48) \gg 2$ , where  $V\{0110\} * 2^3 = 48$

y2	0	1
symbol_locate	S3	Goto table3

Sectioning the tree at  $L=7$ , gives  $y = (x-52) \gg 0$ , where  $V\{01101\} * 2^2 = 52$

y3	0	1	2	3
symbol_locate	S4	S5	S6	S6

15

Although the present invention has been described with reference to various particular examples, it should be understood the various variations and modification can be made without departing from the spirit and scope of the invention.